Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Technology 5 (2012) 500 – 508

Procedia Technology

CENTERIS 2012 - Conference on ENTERprise Information Systems / HCIST 2012 - International Conference on Health and Social Care Information Systems and Technologies

Ubiquitous Integration Architectural Issues

João Henriques[a], Paulo Tomé[a,b,*]

[a]*Instituto Politécnico de Viseu, Departamento de Informática, Viseu, Portugal*
[b]*Centro de Investigação Algoritmi*

Abstract

The paradigm of heterogeneous systems has been demanding the implementation of new approaches to ensure the organizational operations as a whole. This work proposes a new integration model, which can be applied to any system, based on the execution of instructions transmitted between systems supported by research in an organization with great needs at integration systems level.

Keywords: Architecture; Integration Model; Interoperability; XML Programming Language.

1. Introduction

Systems integration is an activity that requires interoperability between different entities, adding value and reducing costs maintenance, and an area that needs research and development [1]. The interoperability process contains various entities that intend to change some information to contribute with benefits to the involved business processes, optimizing its activities in the context of organizations, including information systems.

* Corresponding author.
E-mail address: ptome@estv.ipv.pt

doi:10.1016/j.protcy.2012.09.055

Moreover, the evolution, migration and integration of the existing software to the software legacy is an enormous challenge to business [2, 3].

Interoperability is the ability of two or more software systems communicate and cooperate between them [2], contrary to the language differences, interfaces or execution platform [4]. Interfaces represent the point at which independent systems or components operate and communicate with each other or the limit by which systems share information [10]. Actual protocols do not offer uniform interfaces across the application space and are thus complicated to integrate with traditional enterprise applications [5].

This paper presents and describes the Ubiquitous Integration model to reduce the difficulties related to the design and implementation of integration solutions. This approach has the intention to streamline and remove the complexity related to the definition and implementation of distributed solutions, enabling different systems to implement this model, allowing a high degree of integration, supported by a common protocol with benefits in terms of reducing complexity, time and implementation costs.

Second section describes the state of art and then third section presents the need of a new integration model. Fourth section describes the Ubiquitous Integration model. Fifth section presents XML programming language. Sixth section details this interoperability process model. Last section presents some conclusions and refers possible paths in the implementation Ubiquitous Integration Model.

2. State of Art

From an enterprise perspective, is made a distinction between application-to-application, business-to-business and business-to-consumer integration. Fundamental integration concepts include Enterprise Integration and Enterprise Service Bus, middleware and messaging [6], to support data transfer.

Most recent integration architectures are based on concepts such as event-driven architecture, grid computing or extreme transaction processing [7].

Integrations levels considered as data level, object level and process level.

Currently, there is a set of base technologies to achieve the integration problem, such as RPC, CORBA, COM, WEB SERVICES, J2EE, .NET and SOA.

Many of these models are proprietary and are defined using common sense and definitions of architectures.

3. New Integration Model Approach

It is intended that the integration solution meets certain aspects, such as bi-directionality, universality, simplicity, configurability, responsiveness, robustness, deployment time, cost reduction and instructions transfer to obtain its adoption by the concerned community.

The integration solution may require low effort as possible, so the architecture must be intuitive and allow them a quick learning curve. These factors will contribute to the adoption of this model.

This new integration model can be applied to any system, present or future, supported by a common model, in which any application can interact with other application, at the application layer, on the same or different machine. This kind of integration model, accessible anytime and anywhere, can be viewed in a ubiquitous way, naming this integration model.

4. Architecture

The components of the proposed architecture of Ubiquitous Integration model are illustrated in Fig. 1. The components of this model are the server adapter, server, client, client adapter, envelope, descriptor, the client and server agent and finally, the interpreter.

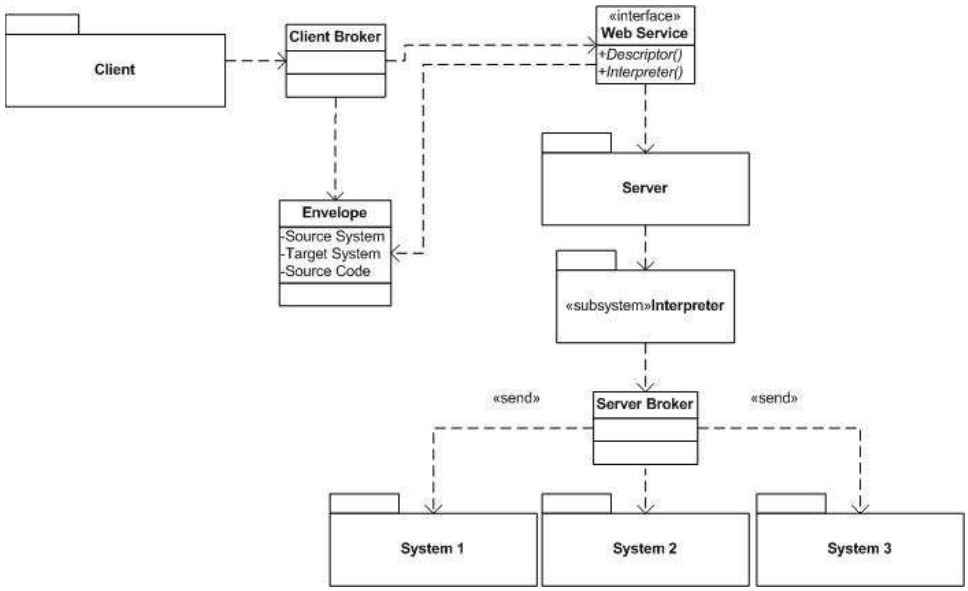

Fig. 1. Ubiquitous Integration Model

It is proposed a new architecture sustaining the interoperability process between different systems, in an environment populated by heterogeneous systems, supported in the use of a free XML programming language, understood by the involved systems, in order to facilitate the expansion of this architecture. To this architecture contributed decisively different integration experiences and concepts provided by SAP R/3.

System Server The server system is responsible for the publishing of the functionalities and provides the mechanism to process the instructions received in a free XML programming language, which the system understands.

The delegated instructions come from the client system. The server receives and processes the orders. The server validates the compliance of sent instructions, in terms of structure of the XML programming language and the correct use of the published library functions.

After instructions interpretation, the result is returned to the client system. Success or errors are returned to the client system.

Server Adapter System. The server system does not have, natively, the features needed by the Ubiquitous Integration model. The integration could be implemented and supported by a server adapter that will carry out the adaptation to this model. Thus, it is not necessary to take changes in the source code of the base system, where the functionalities are located.

*Client System***.** The client system applications access the server systems, using the features published by them. This model does not require local validation of instructions to delegate to the server system. This validation is performed by the server system. The client system must know the programming language based on XML, to structure the instructions sets. The client system must also need to know, in advance, the features published by the server system.

Client Adapter System. If a client system cannot organize the instructions on the free language in XML format and have the functionality natively access Ubiquitous Integration Model, then it must be implemented

an auxiliary system that will perform the adaptation to this model. Thus, it is necessary to intervene on the client system.

Envelope. It is understood by all the agents on this model that the envelope contains all information, the system emitter, the role of the client, the system receptor, the role of server, and the message, identifying the programming language instruction that supports the communication between the two systems. The credentials must also be part of the envelope. Fig. 2 shows an example of this type of envelope, with all contained elements.

```
<envelope>
<id>46195</id>
<user>jhenriques</user>
<password>pas123w4d</password>
<source>192.168.1.12</source>
<sourceport>3220</sourceport>
<destination>192.168.3.56</destination>
<destinationport>3220</destinationport>
<date>20110831</date>
<time>22:43:12</time>
<language>O:XML</language>
<instructions>
<o:type name="Test">
<o:variable name="input"/>
<!-- test input vector -->
<o:variable name="result"/>
<!-- expected test result -->
<o:function name="Test">
        <o:param name="input"/>
        <o:param name="result"/>
<o:do/>
</o:function>
<o:function name="run">
<o:do>
        <o:set ret="$this.test($input)"/>
        <o:assert test="$ret = $result"/>
</o:do>
</o:function>
</o:type>
</instructions>
</envelope>
```

Fig. 2. Ubiquitous Integration Envelope

Descriptor System. This component describes the system, with detailed features allowed by the system that implements the Ubiquitous Integration model. Through a description mechanism that is located in an independent archive, to list the classes and methods, input parameters, result and corresponding type. The description of the available features in the system is made by an archive, listing then classes and methods available to other systems.

Client Broker. Is a client component with the task of providing the channel to send the envelope, containing the instructions to be processed by the server. This interface is responsible for compression and encryption of data functionalities, before transmitting them to the server system and also by the use of credentials. In this component is possible to intervene in order to ensure efficiency and safety of the use of the transmission channels.

Server Broker. Is a component server that is responsible for the receptor channel and for the entry of the envelope received from the client system, to be internally processed. It is also responsible for the complementary activities of credentials validation, decompression and decryption of data received from the client system and contributes also for efficiency and safety involved in the use of communication channel.

Interpreter. The interpreter is located on the server system. This main component is responsible for the interpretation of the set of instructions contained in the message envelope, delegated from the client system.

5. XML Programming Language

Technologies play an important role in how the architecture is implemented and contributes to an ideal solution to solve relevant problems. This model suggests the use of an adapted programming language, supporting the communication between heterogeneous systems.

The use of a XML programming language meets the requirements of this model, such as easiness of understanding, allowing the implementation of programming models and object-oriented paradigms that can easily be transported through Internet channels, based on HTTP and SOAP technologies.

The language allows the parsing of the source code, using validators. This validation is performed at two levels, firstly, the tags and text parsing, and a secondly for the interpretation and validation of the structure of these tags with schema. A XML programming language can take advantage of these capabilities, provided by many available text editors.

The verbosity of XML representation can be addressed through functionalities that convert the XML into a simplified language version, more legible and adapted to human understanding.

It is not advocated the use of just one XML programming language, in particular, because that can exist other languages that could comply the principles of this model, such as the existing ones, SuperX++ [7] and o: XML [8] languages.

6. Interoperability Process

The interoperability process, under this Ubiquitous Integration model needs a set of steps, like structuring, sending, receiving and interpreting the requested instructions.

The initial structuring phase is triggered by a particular client system need, to obtain a set of results from another server system. Firstly, is necessary to properly structure an instruction set, organizing this information to be delegated to the receptor system. Thus, the client system must organize an order through an envelope, referring the programming language of the XML instructions, assembly instructions and information identifying the client and server system, and the corresponding user credentials. The envelope has all the elements, enabling the execution of the target system instruction set.

The client system, after knowing the available features in the server system, prepares and sends an envelope to the system server, containing the instructions. It is now possible to proceed with the client system and send out those instructions to a second application server. The target system waits for the client requests. Once approved such application is validated and forwarded to the internal parser. The request received in the server system also generates a sequence number for the identification and tracking of the integration process.

The interpretation phase processes the client system delegated instruction and this instructions are mapped to the local application, which will perform its processing.

Local interpretation, on the server system, largely depends on its platform. At this point it is necessary to forward the request to place the interpreter, together with the instructions contained in the envelope. This process is performed with the reception of the envelope in the receptor system.

Local interpreter can be configured with a local structured archive. The service for the interpretation of the instructions is configured through an archive in XML format. This archive contains the location of the responsible component for the interpretation of the delegated instructions. May also be available various interpreters in the same system. This will enable the extension of the base capacities related to the design of integration solutions.

After the envelope reception, the server system assigns an order number, identifying the client system request.

The interpreter represents the implementation core of this architecture meeting a specific interface, so that it can be implemented in different ways, while maintaining the homogeneity of the integration model.

The interpreter executes local interpretation of the delegated instructions, mapping the classes and methods published by the server system to internal features, located on hosted applications.

The classes and methods are mapped to libraries and applications by a configuration archive, also in XML format. This archive describes the class and method and published names of the parameters, mapping these ones to the hosted functions located on libraries.

After server system processing this set of delegated instructions, and more specifically the local interpreter, the response is collected to an envelope and is returned to the client system.

Fig. 3 illustrates main activities involved in processing a request under Ubiquitous Integration model and Fig. 4 presents the main elements of this architecture.

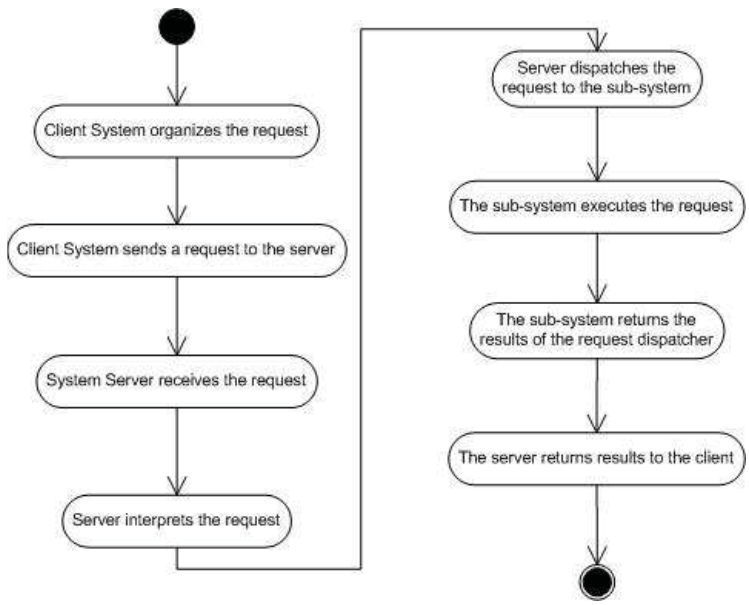

Fig. 3. Activity Diagram

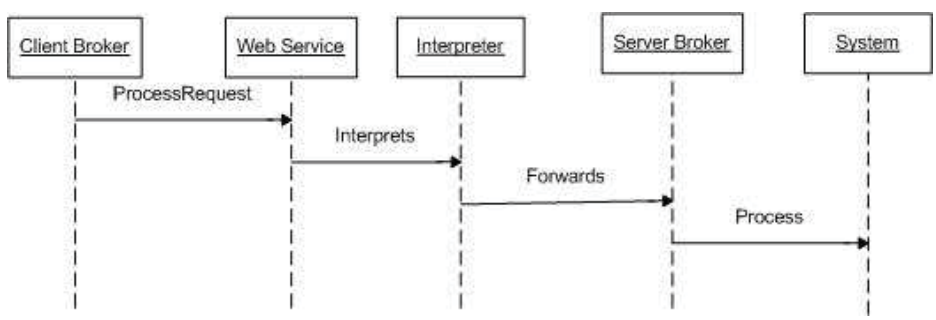

Fig. 4. Sequence diagram

6.1 Integration Web Service

The implementation of this architecture requires a web service to support integration activities. This web service must provide two essential methods, allowing the understanding of the features provided by the server system, in order to structure client requests.

Every system can interact with one another, without local software and a set of specific libraries. This type of integration can be used on the development of one specific integration solution, where two independent systems exchange any kind of information, data or instructions.

A web service is a well-known technology, present in most IT departments and a technology embraced by virtually all platform providers, proven technology, open and platform-independent. The SOA (Service Oriented Architecture) concept has proven very successful for gluing heterogeneous systems [5, 9].

The server component lists his functionalities, receives external requests and executes them, returning the corresponding results to the client system. To this purpose, the web service must contain two distinct methods, to grant the integration process. The first method will be responsible by the interpretation of external requests and a second, to list and describe the server features. Additionally to these methods, there´s also the descriptor of the web service, WSDL. All systems that intend the implementation of the Ubiquitous Integration model must provide these two web methods, one to list and describe the features offered by the server and another method for the reception and interpretation of the instruction set delegated in XML format. After the interpretation of delegated instructions, the system returns a result set to the client system, also structured in XML format.

Interpreter. The web service, have published a method, named Interpreter, responsible for the interpretation of the instruction assigned by the server system. The instructions are organized in XML format. This method is responsible for forwarding the delegated instructions to an internal interpreter, configured through an archive in XML format. Any internal application could be used as an interpreter, located in a particular platform, receiving instructions and a second application, can provide the result set. These results will be collected by the client system for further actions.

*Descriptor***.** This method publishes the functionalities on the server system to the client systems. Through a XML format archive, the information is available to the client systems with the composition of its functionalities. This archive will provide a set of classes and methods that forms the library to provide functionalities to the client systems, who wants to use them.

6.2 Calculator Example

As a simplest example of the possibilities of the use of Ubiquitous Integration model, describing a typical application calculator providing the arithmetic capabilities and allowing any client application to invoke those main server functionalities to calculate the product of prime numbers.

Based on this integration model can be available to the client systems the features of the calculator application, in a way where not only the functions are invocated, but also the server receives the instruction set or a complete program, that uses the functions provided by the server system. The transmission of the instruction differs from the unitary use of a web method.

The web method “Descriptor”, of the web service, describes the TCalculator class, with methods to add, subtract, multiply, divide, and clear the results, according to the buttons of the user interface of this type of application.

Any client system that wants to use these calculation capabilities of this application does not need to have the application installed locally, nor implement a complicated existing integration model, neither develop an

independent project to address the proposed objectives. It is sufficient, to the client system, arrange a set of instructions that, flowing its pretended logic, firstly, instantiates an object of the class TCalculator and then relying e.g. “multiply” method, using two values, according to the parameters required by this method, returning the result of this method to a variable with the result. The result of this calculation can be placed in the output of the program.

After the client system has, locally, organized the instructions in a program to calculate the product of prime numbers, they are transferred to the server system, under a web method “Interpreter” of its web service.

The server system collects the instructions and verifies used programming language, forwarding these ones to the corresponding internal interpreter. This component performs the verification and validation of the instructions, parsing and anticipating compilation errors. After this step, the server system interprets these instructions, checking the source code and the classes and methods used. In this case is used TCalculator class and “multiply” method. It is checked the configuration archives of the library responsible for this class and mapping the methods to the internal calculator.dll, containing the several base arithmetic functions. The web method “Interpreter” maps the specific functions to the hosted DLL, taking advantage of its calculation capabilities, collecting the results, putting it in a return variable. In the last step the set of results are provided into the variable in text format under the output buffer.

After the instructions interpretation, the server system returns the entire contents of the output buffer to the client system, the results of the instructions interpretation. The client system will collect the results of the calculation and perform subsequent operations, related to the client system flow logic program.

In this example the instructions set are, primary, organized by the client system and then transferred to the server system, supported by the use of web services, complementing of the simple unitary invocation of the web methods.

7. Conclusions and Future Work

Based on the solutions done and lessons learned by the experiences in the integration implementation area and using existing technologies, is proposed a new model, named Ubiquitous Integration, allowing the adaptation or implementation of existing system under this model, supported by the transmission of XML instructions, organized and collected by the client system and executed by the server.

Main benefits and differences to the state of art resides in taking advantage from web services and SOAP messages, to provide the channel to grant the mechanism of transferring instructions and collection of results, based on a simple and a free XML programming language, contrary to the simple actual data-centric transfer approach or models and restricted programming languages used. Instructions and data will be simultaneously present in the same message, allowing the evolution and scalability of the systems without intervention on his core source code. This model allows too, the development of solutions without the need of recurring to a local interpreter, because the instructions are interpreted at the server side.

The limitations of this model can be related to the number and volume of instructions that can be transferred between systems and generated traffic under the network and the duplication of the instructions in several requests sequences.

Future work should be related to the analysis of main advantages, alternatives and limitations of this model and also, the implementation of the proposed model to a real system and situation, analyzing the work done.

References

[1] Land, R., 2006. Sofware Systems in-house integration: Observations and Guidelines Concerning Architecture and Process (Phd Thesis).

[2] Bachmann, F., Bass, L., Buhman, C., Comella-Dorda, S., Long, F., Robert, J.E., Seacord, R.C., Wallnau, K.C., 2000. Volume II: Technical Concepts of Component-Based Software Engineering. CMU/SEI.
[3] Britton, C., Bye, P., 2004. IT Architectures and Middleware: Strategies for Building Large, Integrated Systems, 2nd ed. Pearson Education.
[4] Wileden, J.C., Kaplan, A., 1999. Software Interoperability: Principles and Practice. Presented at the 21st International Conference on Software Engineering, ACM, pp. 675–676
[5] Karnouskos, S., Guinard, D., Savio, D., Spiess, P., Baecker, O., Trifa, V., de Souza, L.M.S., 2009. Towards the Real-Time Enterprise: Service-based Integration of Heterogeneous SOA-ready Industrial Devices with Enterprise Applications, in: 13th IFAC Symposium on Information Control Problems in Manufacturing (INCOM), Moscow, Russia.
[6] Gaur, H., Todd, D., Poduval, A., 2012. Do more with SOA Integration: Best of Packt book. Kobo.
[7] Mati, K., 2007. Introduction to x++ [WWW Document]. URL http://www.perfectxml.com/content.asp?ct=xpp
[8] Klang, M., 2004. Introducing o:XML [WWW Document]. URL http://www.xml.com/pub/a/2004/07/21/oxml.html?page=2
[9] Grady, J.O., 1994. Systems Integration. CRC-Press.
[10] Wegner, P., 1996. Interoperability. ACM Comput. Surv. 285–287.